Datorseendebaserad tolkning av teckenspråk

En studie av tidigare metoder och implementering för det svenska handalfabetet

Lukas Lönnroth

Examensarbete
Informationsteknik
2020

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	7622
Författare:	Lukas Lönnroth
Arbetets namn:	Datorseendebaserad tolkning av teckenspråk
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	Yrkeshögskolan Arcada

Sammandrag:
För att icke-talförmögna personer ska kunna kommunicera krävs det oftast att den som de kommunicerar med kan teckenspråk. Ändå är det ganska få i dagens läge som kan kommunicera med teckenspråk. Det här arbetet beskriver de tillvägagångssätt som använts för att försöka bryta den barriären. Inledningsvis har en kritisk litteraturstudie gjorts av tidigare forskning inom området, arbetet lyfter fram för och nackdelar för dessa samt hur de skulle kunna tänkas användas. Därefter har en datainsamling utförts med hjälp av vilken en ny korpus för det svenska handalfabetet kunnat skapas. Med de två tidigare metoderna som grund har sedan fyra olika maskininlärningsmodeller som baserar sig på ett faltningsnätverk skapats. Modellerna har evaluerats och den modellen som klarat sig bäst och på träningsdata uppnått en precision på ca 96 % har sedan använts för att skapa en datorseendebaserad tolk som i realtid översätter de tecken som den presenteras med.

Nyckelord:	Datorseende, djupinlärning, neuralt-nätverk, faltningsnätverk, konvolutionsnätverk
Sidantal:	34
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	7622
Author:	Lukas Lönnroth
Title:	Computer vision-based sign language interpretation
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	Arcada University of Applied Sciences

Abstract:
In order for non-speech persons to be able to communicate, it is usually required that the person with whom they are communicating knows sign language, yet there are relatively few persons who are able to understand sign language in this day and age. This work describes the methods used to try to break that barrier. To begin with, a critical literature review has been conducted on previous research within the field. The review highlights the advantages and disadvantages of the methods used and whether they would fit for this specific implementation. Subsequently, a data collection has been carried out with the help of which a new corpus for the Swedish hand alphabet has been created. With the aforementioned methods as the basis, four different machine learning models based on a convolutional neural network structure where created. The models have been evaluated and the model with the best performance and an accuracy of about 96 % on training data has then been used to create a computer vision-based interpreter that is able to translate signs of the Swedish hand alphabet in real-time.

Keywords:	Computer vision, deep learning, neural network, convolutional neural network
Number of pages:	34
Language:	Swedish
Date of acceptance:	

INNEHÅLL

FIGURER OCH TABELLER

1 INLEDNING

1.1 Bakgrund

Synen är ett av de viktigaste sinnen som människan har och upp till 80 % av alla intryck vi får kommer via den (Low vision international, 2020). Vår värld är uppbyggd efter synen och den krävs för att vi lättast ska kunna ta oss fram. För att datorer ska kunna tillämpas i en sådan värld krävs datorseende. Ett tillämpningsområde för detta är tolkning av teckenspråk.

För att kunna kommunicera med andra personer använder hörselskadade, döva och icke-talförmögna personer sig ofta av teckenspråk. Det förutsätter dock att båda parterna förstår teckenspråk eller att det finns en tolk på plats. Det finns inget universellt teckenspråk (Pulkkis, et al., 2019) utan ungefär 160 olika teckenspråk (Institutet för språk och folkminnen, 2014). Det betyder att även svenska har ett eget teckenspråk som skiljer sig från de andra. Det finns även ett finlandssvenskt teckenspråk som numera bara ca 90 döva talar (Språkinstitutet). Därför kommer det här arbetet att fokusera på det svenska teckenspråket SSL vilket ca 30 000 talar (Sveriges Dövas Riksförbund, 2019).

1.2 Syfte och mål

Att kunna tolka teckenspråk med hjälp av datorseende är ett ämne som det redan finns en hel del forskning om (Keskin, et al., 2012). Däremot finns det ännu inte någon implementering av en tolk för SSL. Syftet med den praktiska delen i det här arbetet har varit att med hjälp av tidigare forskning bygga en tolk som kan urskilja det svenska teckenspråksalfabetet från A till Z (Å, Ä och Ö inkluderas ej då dessa är rörliga och kräver ett annat tillvägagångssätt) med en säkerhet på 80%. En människa ska kunna sitta framför en kamera och visa olika tecken som tolken sedan skriver ut på skärmen.

Syftet med den teoretiska delen har varit att ge läsaren en överblick över existerande teknologi som lämpar sig att användas för ändamålet och tidigare forskning inom området. Till den teoretiska delen hör också en beskrivning över hur implementeringen av den praktiska delen gått till.

1.3 Metoder

Innan implementeringen har en kritisk litteraturstudie inom området utförts så att jag sedan på basis av den kunnat ta informerade beslut då planering och implementering av arbetet utförts.

För implementeringen av den praktiska delen har verktyg så som Python, Tensorflow, Google Colab och OpenCV använts. En grundligare genomgång om varför just de verktygen använts hittas i implementeringsdelen av arbetet.

1.4 Arbetets struktur

Till en början kommer jag att gå igenom tidigare metoder som använts dels för handigenkänning och dels för tolkning av handrörelser. I detta skede tas det upp för- och nackdelar med de olika metoder som använts.

I implementeringsdelen av arbetet presenterar jag de tillvägagångssätt som använts för att uppnå målet. En beskrivning av datainsamlingen samt strukturen på insamlad data. Implementeringen baserar sig på jämförelse av tidigare forskning.

Avslutningsvis gör jag en genomgång och en kritisk analys av huruvida implementeringen har uppnått de angivna målen för arbetet. Därefter följer en slutdiskussion om arbetet och resultaten samt hur det skulle kunna förbättras i framtiden.

1.5 Avgränsning

Arbetet behandlar inte ingående hur neurala nätverk och faltningsnätverk fungerar och alla de olika formerna de kan existera i, då det redan finns en stor mängd resurser online där man kan lära sig om djupinlärning och alla dess komponenter. I arbetet presenteras de tillvägagångssätt som använts just för den här implementeringen och tidigare forskning relevant för den.

2 TIDIGARE FORSKNING

I det här kapitlet ges en överblick av tidigare forskning som gjorts för att urskilja objekt från bakgrunder samt igenkänning av rörelser.

2.1 Handsegmentering

För att minska på mängden data en modell måste bearbeta och därmed minska på träningstid samt resursanvändning kan man först känna igen var på bilden det finns en hand. Då kan man isolera den delen av bilden så att modellen har en mindre bild att arbeta med. Detta brukar kallas ”*Region of Interest”* (ROI) eller intresseregion.

2.1.1 Förbehandling av bilder

Förbehandling av bilder används oftast inom datorseende eftersom det blir lättare för en modell att identifiera vad som händer på bilden om den ser ut enligt en viss standard. De vanligaste sätten att standardisera bilder på är bl.a. genom att se till att alla bilder har samma storlek eller att se till att alla har samma färg eller gråskala. Bilder konverteras ofta till gråskala för att spara på resurser då en gråskalig bild bara innehåller en kanal, istället för tre kanaler som behövs i färgbilder.

2.1.2 Segmentering med hjälp av hudfärg

Segmentering med hjälp av hudfärg innebär att man analyserar bilden och anser allt som inte matchar en hudfärg vara bakgrund. Så kan man urskilja t.ex. en hand från bakgrunden på bilden. Detta kan uppnås med ett antal olika metoder. Figur 1 illustrerar segmentering med hjälp av hudfärg.

Figur 1 Handsegmentering med hjälp av hudfärg.

Enkel Gaussisk modell

Den Gaussiska modellen är välanvänd inom datorseende och fungerar genom att jämföra varje pixel med modellen. Om pixeln matchar modellen så anser man den som hudfärg. (Wang & Pan, 2012) Modellen fungerar dock dåligt då hudfärgen på handen eller ljusstyrkan varierar. (Wang & Pan, 2012) Detta kan man kringgå genom att använda sig av en enfärgad handske då man tecknar framför kameran som Nagi (2011) gjort eller att använda sig av en annan form av modellen som mäter färgen i mitten av handen och därmed kunna uppdatera de parametrar som modellen använder sig av. Då får man en ständigt uppdaterad modell som fungerar under flera möjliga omständigheter.

2.1.3 Bakgrundssegmentering med hjälp av medelvärde

Ett resurseffektivt sätt för att segmentera bort bakgrunden för en bild är att använda en metod som först räknar ut ett medelvärde för hur bakgrunden i bilden ser ut innan man introducerar objektet som man vill urskilja. Något som Ilango (2017) har gjort. Man använder sig sedan av ett gränsvärde för att skilja på de olika färgerna i bilden för att sedan göra sig av med bakgrunden. Detta medför dock att man måste använda sig av en bild på endast bakgrunden och sedan en bild med objektet eller en videoström där man väntar en stund innan objektet introduceras. Bakgrunden bör också vara så enfärgad som möjligt för att den här metoden ska vara effektiv. I figur 2 kan man tydligt se hyllan med saker komma fram i handen.

Figur 2 Segmentering med hjälp av medelvärde. En jämförelse mellan en tom bakgrund och en bakgrund med bråte.

För något så statiskt som en webbkamera kan detta ändå vara ett bra tillvägagångssätt då bakgrunden oftast hålls konstant då man väl börjat använda kameran.

2.1.4 Binär färgskala

I motsats till bilder i gråskala där pixlar kan ha ett varierande värde kan bilder i binär färgskala bara innehålla ett av två möjliga värden. Rahaman et al. (2014) använder sig av just binär färgskala för att känna igen tecken ur Bengali-teckenspråket. En nackdel med användningen är att allt framför handen inte går att urskilja. Ett tecken måste alltså kännas igen på dess kontur och inte något som utmärker det i själva tecknet. Ett exempel på det här finns i figur 3. I färgbilderna ser man lätt vilken skillnaden är mellan tecknen men då man använder sig av en binär färgskala blir skillnaden inte lika tydlig. För det svenska handalfabetet skulle det här inte inverka så mycket eftersom alla tecken tecknas med en hand och det sällan skulle uppstå liknande situationer som finns i figur 3. Däremot om man vill utöka arbetet till att börja tolka teckenspråk skulle det snabbt uppstå liknande situationer.

Figur 3 Nackdel med användning av binär färgskala.

2.1.5 Random forest

Random forest (RF) är en form av klassificeringsalgoritm som består av en sammansättning av flera beslutsträd, där varje beslutsträd tar ett beslut och det beslutet som sedan har flest ”röster” blir det slutgiltiga beslutet (Yiu, 2019). RF har bland tidigare forskning använts ett flertal gånger för att identifiera händer eller tecken och har visat sig vara ett effektivt sätt att urskilja händer från bakgrunder. Man har med hjälp av RF kunnat hitta händer på bilder med en precision på runt 90% (Sangjun, et al., 2015).

Multi-layered random forest (MLRF)

Kuznetsova et al. (2013) föreslår att använda sig av en MLRF för att tolka teckenspråk. Detta för att potentiellt nå en högre säkerhet och kortare träningstid samt mindre resursanvändning.

I jämförelse med RF så kan man minska träningstiden med ca 90 % utan att försämra precisionen eller till och med höja den genom att använda sig av MLRF (Kuznetsova, et al., 2013).

2.1.6 ”Haar-cascade” baserad handidentifiering

Haar-cascade klassificering är en maskininlärningsbaserad metod för att identifiera objekt i bilder introducerad av Viola och Jones (2001). Man använder då flera simplare eller svagare klassificerare för att snabbare kunna urskilja objektet man söker från bakgrunden. Metoden användes ursprungligen för att kunna urskilja ansikten på bilder. Men den har även använts för händer (Rahaman, et al., 2014).

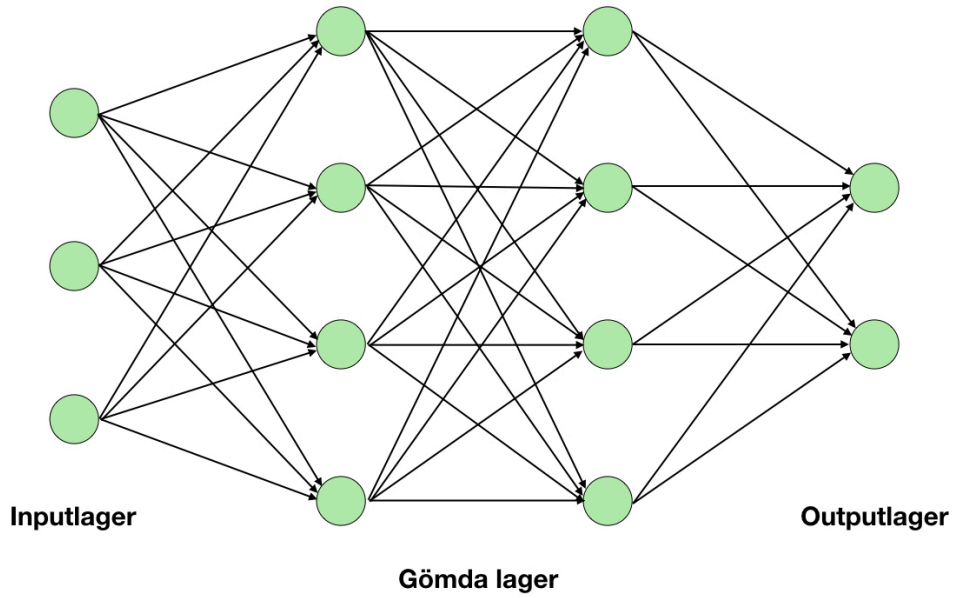
En haar-cascade modell tränas på positiva och negativa bilder dvs. bilder som innehåller objektet modellen ska känna igen och bilder som inte innehåller det. Modellen går igenom bilden och extraherar så kallade haar-särdrag som sedan används för klassificering. Haar-särdragen är simpla särdrag som används för att hitta kanter och linjer.

Fördelen med att använda sig av metoden är att den är oberoende av hudfärg (Viola & Jones, 2001) och resurseffektivare än andra metoder som t.ex. djupa neurala nätverk. En nackdel med metoden är dock att då man använder den för objekt som kan förekomma i flera olika stadier, t.ex. öppen hand eller stängd hand, är att modellen då måste tränas för alla de olika stadierna.

2.2 Neurala nätverk

Neurala nätverk (NN) eller artificiella neurala nätverk (ANN) är modeller inspirerade av de neurala nätverken man hittar i människor och djur för att känna igen mönster. De består av många små enheter som arbetar parallellt utan någon centraliserad styrenhet (Patterson

& Gibson, 2017, s. 263). Nätverken fungerar genom att de har ett inputlager och ett outputlager. Mellan dessa finns det sedan n stycken gömda lager. Vikterna för kopplingarna mellan neuronerna uppdateras sedan under träning. På det sättet kan nätverket nå bättre resultat. Figur 4 illustrerar hur ett simpelt neuralt nätverk kan se ut.



Figur 4 Illustration av ett neuralt nätverk.

En form av neurala nätverk är så kallade faltningsnätverk (Convolutional neural network, CNN). Dessa brukar räknas som djupa neurala nätverk och används ofta inom datorseende tack vare deras förmåga att upptäcka mönster som framkommer på olika platser i bilder. Då man använder sig av faltningsnätverk blir antalet vikter i nätverket betydligt färre än i ett traditionellt neuralt nätverk med samma antal lager (Stanford).

2.2.1 Faltningsnätverk för tolkning av teckenspråk

Rao et al. (2018) föreslår att använda sig av ett faltningsnätverk för att tolka teckenspråk. Med sin modell lyckas de på sin testdata få ett imponerande medeltal av igenkänning på hela 92,88% vilket slår tidigare toppmoderna modellers resultat på samma data. Deras CNN är också betydligt snabbare än de andra metoderna på att klassificera de olika tecknen även om det tar något längre tid att träna det. Faltningsnätverket som användes var inspirerat av Krizhevsky et al. (2012) och bestod av fyra faltningslager följt av tre klassificeringslager. Att notera är att man i den här forskningen inte bara klassificerade enstaka bilder utan hela rörelser.

Av de metoder som använts i tidigare forskning verkar faltningsnätverk vara den som lämpar sig bäst för ändamålet. Implementeringar för andra språk har visat sig fungera bra och kunnat tolka tecken med hög precision. Vad gäller bildhanteringen finns det lite olika komplikationer med de olika tillvägagångssätten. Användningen av binär färgskala eller bakgrundssegmentering med hjälp av medelvärde medför de tidigare konstaterade nackdelarna. För att undvika de här nackdelarna prövas användningen av kantupptäckning i implementeringen. En metod som även tidigare använts inom datorseende och går ut på att hitta gränser mellan objekt i bilden.

3 IMPLEMENTERING

Implementeringen består huvudsakligen av tre moment: datainsamling, byggandet av modeller och evaluering av modeller. I det här kapitlet presenteras vilka olika delmoment som ingår i de olika faserna och hur de har förverkligats. Kapitlet presenterar också i korthet vilka verktyg som använts.

3.1 Verktyg

Nedan beskrivs de verktyg som använts samt en förklaring varför det lönar sig att använda dem för just den här implementeringen. Programmeringsspråket som använts är Python 3.7 och den huvudsakliga texteditorn är Vim.

OpenCV

OpenCV är ett bibliotek med öppen källkod för datorseende och maskininlärning som innehåller en mängd olika algoritmer för bildhantering och maskininlärning. Biblioteket innehåller också funktioner som gör det enkelt att från ett python-program kunna anropa enhetens webbkamera och utföra olika bildjusteringar på de bilder som webbkameran tar.

Eftersom OpenCV innehåller funktioner som kantupptäckning, bildbeskärning och verktyg för att bygga simpla användargränssnitt för att visa upp webbkameran så lämpar det sig bra för implementeringen av det här arbetet.

Tensorflow

Tensorflow är en plattform för maskininlärning som gör det lättare att t.ex. i python kunna bygga djupinlärningsmodeller då man inte behöver koda modellen från början, utan man kan använda sig av färdigbyggda funktioner för att på en abstraktare nivå definiera sin modell.

För det här arbetet lämpar sig Tensorflow utmärkt, eftersom man med hjälp av plattformen kan fokusera på att bygga modeller som producerar resultat, snarare än att koda modeller och lager från noll.

Google Colab

Google Colab är en molntjänst som låter användaren exekvera sin kod på en server istället för på sin egen dator genom så kallade Notebooks. På det här sättet kan vem som helst med tillgång till internet utföra resurskrävande uppgifter såsom djupinlärning.

Tjänsten tillåter användaren att koppla sin Google Drive lagringstjänst direkt till sin notebook för att enkelt ha åtkomst till den data man behöver för sitt projekt. Eftersom det enda som fanns tillgängligt i början av projektet var en bärbar dator utan en dedikerad grafisk processor lämpade sig verktyget för projektet då det drog ner träningstiden för modellerna från ca 2 ½ timme till ca 2 minuter.

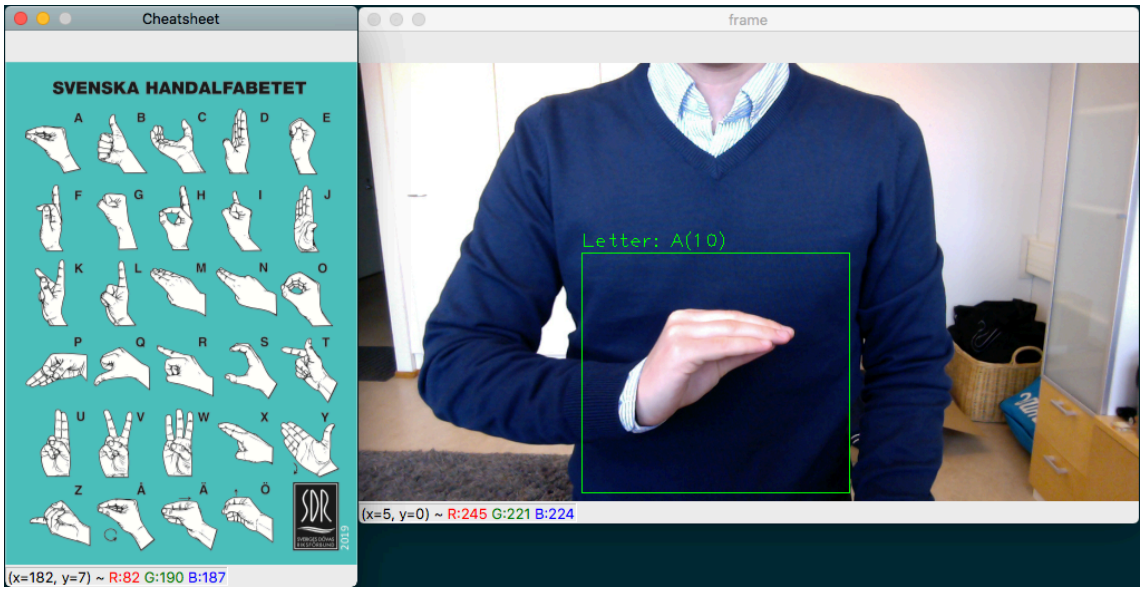
Det finns många olika verktyg för maskininlärning och djupinlärning på marknaden. Det finns inget som säger att just de verktyg som använts för det här arbetet måste användas för något annat liknande projekt.

3.2 Data

Eftersom det för SSL finns en begränsad mängd färdig data för den här typen av projekt så har det för det här arbetet skapats en helt ny korpus för ändamålet. I det här avsnittet förklaras hur insamlingen av data gått till och hur korpusen är strukturerad.

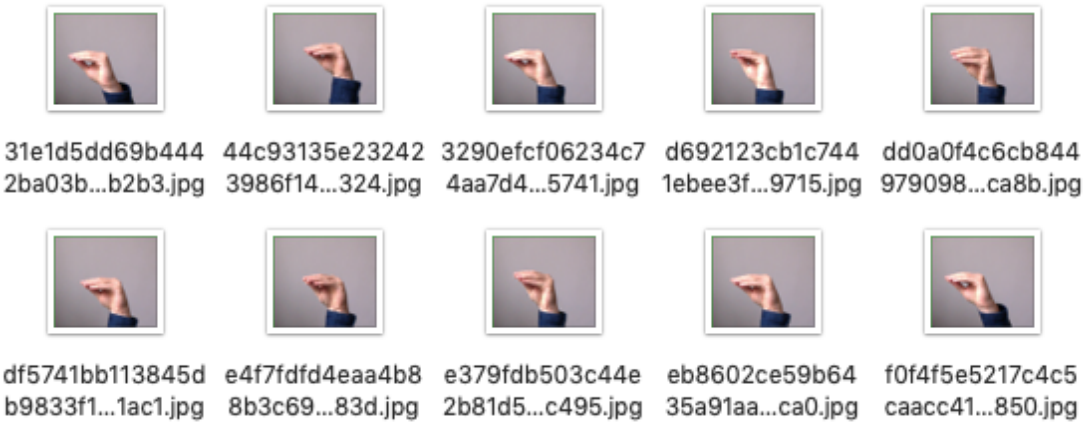
3.2.1 Insamling av data

För att kunna samla in en stor mängd data på möjligast kortast tid användes ett skript där personen som ska teckna presenteras med två rutor på skärmen. Den ena innehåller SSL alfabetet och den andra en videoström från datorns webbkamera. I rutan med webbkameran finns det utritat ett område i vilket användaren ska ha sin hand dvs. ROI, samt vilken bokstav användaren ska teckna. Då användaren anser att tecknet ser bra ut kan den trycka på en knapp för att spara bilden. Programmet sparar automatiskt bilden i en mappstruktur så att alla bilder sparas i den mapp som hör till bokstaven. För varje bokstav uppmanas användaren att spara 10 bilder. Figur 5 visar användargränssnittet som använts.



Figur 5 Användargränssnittet för datainsamling.

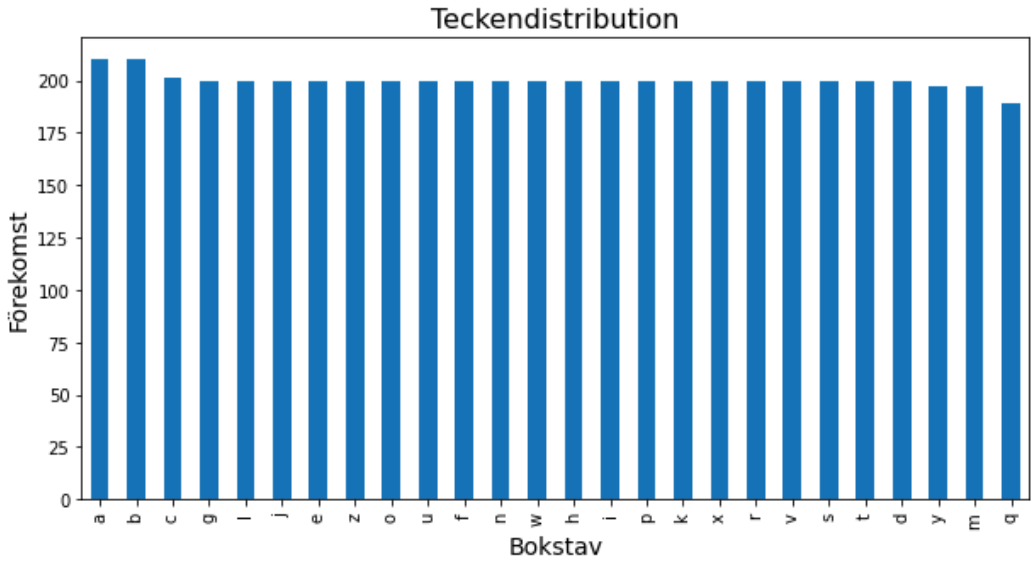
Med hjälp av det här skriptet går det snabbt att generera träningsdata åt modeller jämfört med att manuellt ta varje bild då det för varje runda skapas 260 bilder. Figur 6 visar bilder för bokstaven A som genererats under en runda av skriptet.



Figur 6 Exempel på genererade bilder för bokstaven A efter en runda för skriptet. Alla bilder blir tilldelade en unik id baserat på klockslaget. Detta så att filerna inte ersätts även om man kör programmet i olika omgångar.

3.2.2 Korpusen

Den slutgiltiga korpusen som skapats består av 5204 bilder på tecknen A-Z i det svenska handalfabetet tecknade av fyra olika tecknare mot olika bakgrunder i varierande ljusförhållanden. Bilderna har storleken 128x128 och är i gråskala.



Figur 7 Distribution av bokstäver i korpusen.

Mängden data per tecken i korpusen skiljer sig något, då det alltid vid datainsamling sker misstag och en del data helt enkelt blir oanvändbart. Det tecken som det finns minst data på är Q med 189 bilder och det som har mest är A med 210 bilder. Så det är en rätt så

jämn fördelning som man kan se i figur 7. Man kan också i ett senare skede se att även om Q har det minsta antal bilder så inverkar det ändå inte på modellernas förmåga att känna igen tecknet.

3.3 Tillvägagångssätt

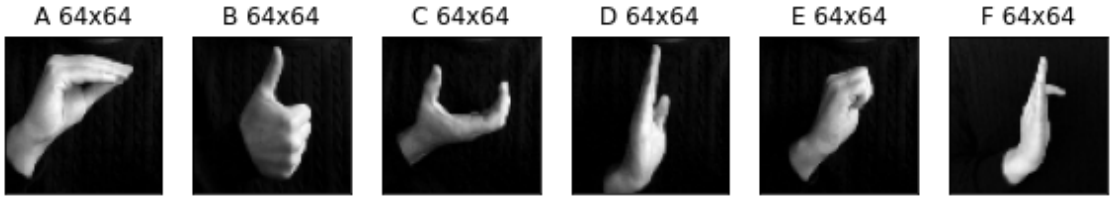
För implementeringen har det gjorts fyra olika modeller som bygger på samma struktur av faltningsnätverk. Det som skiljer dessa modeller är storleken på bilden som modellen tar in och om den använder sig av kantupptäckning eller ej. Det har alltså skapats två modeller som tar in en bild av storleken 64x64, en som tar in en gråskalig bild och en som tar in en bild som det utförts kantupptäckning på samt två modeller som tar in en bild av storleken 128x128. Modellerna kommer att refereras till med namnen Canny_64, Gray_64, Canny_128 och Gray_128.

Tolken som skapats fungerar genom att användaren ser sin egen webbkamera och på den ritas det ut en ruta i vilken användaren ska teckna. Rutans placering är sådan att den som tecknar ska kunna göra det på ett så naturligt sätt som möjligt dvs. lite ovanför midjehöjd. Ovanför rutan skrivs det ut vilken bokstav som tolken anser sig se i rutan.

3.3.1 Databehandling

Databehandlingen är en väsentlig del av maskininlärningsprocessen och bör därför utföras noggrant för att uppnå önskade resultat. Till att börja med delas korpusen upp i tränings- och testdata. Då kan man vara säker på att modellen som tränas aldrig sett den data man testar den med tidigare. För den här implementeringen delades korpusen upp så att 80% av den blev träningsdata och 20% testdata. Viktigt att tänka på i det här skedet är att man inte tar 80% från början och 20% från slutet utan att man försöker jämna ut fördelningen mellan alla tecken. Om man inte jämnar ut fördelningen resulterar det i att modellen endast kommer att kunna känna igen de tecknen som förekommer i testdatan dvs. 80% av korpusen. I det här fallet betyder det att modellen endast tränas på tecknen A-U och testas på tecknen V-Z. Genom att fördela tecknen jämt över korpusen kan man vara säker på modellen klarar av att klassificera alla tecken.

För den här implementeringen användes huvudsakligen två olika metoder för att behandla bilderna. Den första är att konvertera bilderna till gråskala och på det viset minska på mängden data som nätverken måste behandla. I figur 8 kan man se ett exempel på bilder som blivit förminskade till 64x64 pixlar och konverterade till gråskala. Även om bilderna är kraftigt förminskade från originalbilden (före förminskningen då bilderna lades till i korpusen) kan man fortfarande lätt urskilja de olika tecknen.



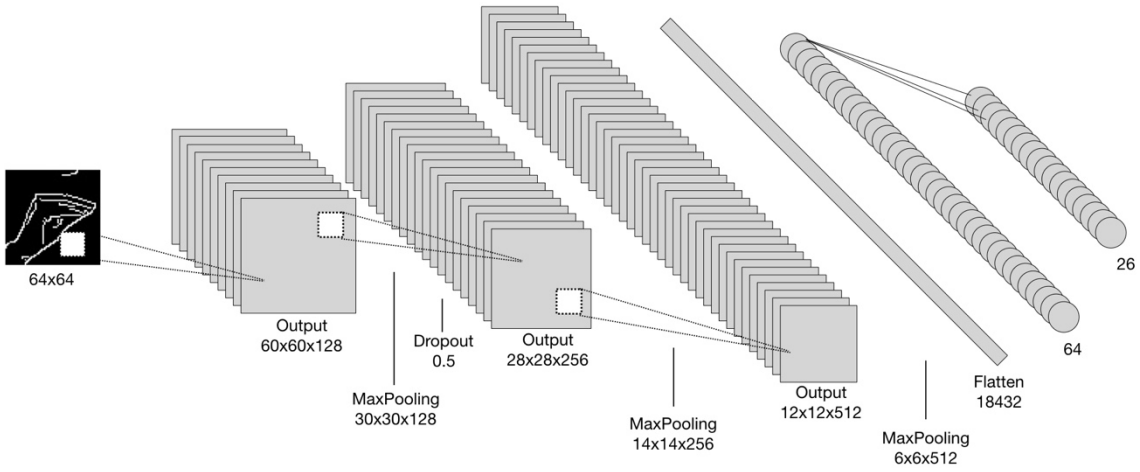
Figur 8 Tecken förminskade och konverterade till gråskala.

Den andra metoden som använts är så kallad kantupptäckning som används för att reducera brus och framhäva kanter i bilden. Kantupptäckningen utfördes med OpenCVs inbyggda ”Canny” funktion som är bibliotekets implementation av John F. Cannys algoritm för kantupptäckning. I figur 9 kan man se samma tecken som i figur 8 med kantupptäckning.

Figur 9 Kantupptäckning med Canny edge detection.

3.4 Nätverkets struktur

Den struktur som slutligen visade sig fungera bäst för det här specifika problemet var ett nätverk med tre faltningslager som skapar 128, 256 och 512 filter följt av ett fullt anslutet lager på 64 noder och ett outputlager på 26 noder dvs. en per bokstav som nätverket ska känna igen. I figur 10 kan man se en illustration av nätverkets struktur och på så vis få en bättre överblick av flödet i nätverket. Längst till vänster ser man bilden som nätverket ser och längst till höger outputlagret.

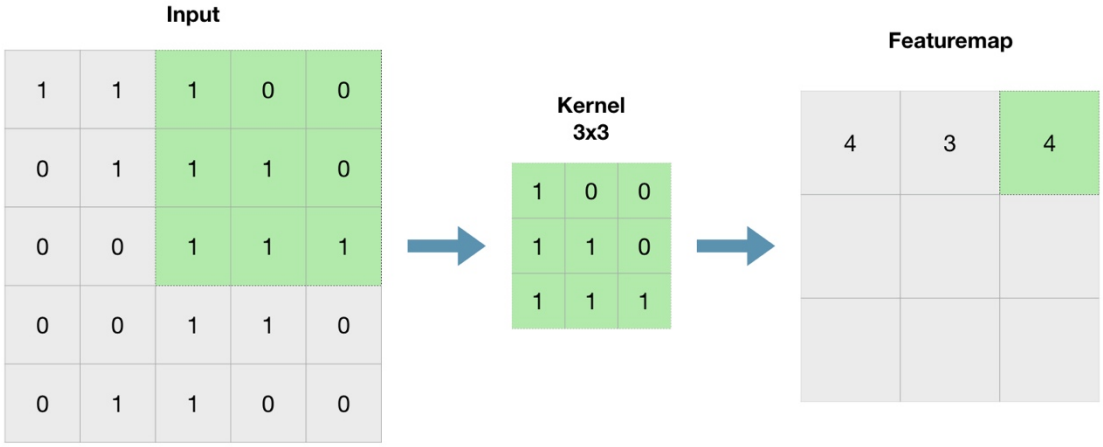


Figur 10 Illustration av nätverkets struktur.

För att det fullt anslutna lagret ska kunna ta emot den output som det sista faltningslagret ger så krävs det också ett utjämningslager (eng. flatten) vars uppgift är att dimensionera ner matrisen som faltningslagret ger ut till ett endimensionellt fält.

3.4.1 Faltningslagren

Då ett faltningslager definieras anges en storlek för filtret som lagret använder sig av för att gå igenom den input det fått. Med hjälp av det filtret skapas det sedan så många särdragskartor (eng. featuremaps) som angetts då lagret definierats. Man anger också en så kallad stride vilket definierar hur mycket filtret ska flytta sig i sidled för varje position. Kartorna skapas genom att filtret räknar skalärprodukten på varje position i bilden som det passerar. Då lyfts särdrag i bilden fram med hjälp av vilka klassificering sedan kan utföras. En illustration av skapandet av en särdragskarta kan ses i figur 11.



Figur 11 Illustration av skapandet av en särdragskarta med ett filter på 3x3 pixlar och stride 1 pixel.

Filterstorlekarna som valts att användas för de tre faltningslagren i nätverket är 5x5, 3x3 och 3x3. För varje lager har en stride på 1 pixel använts.

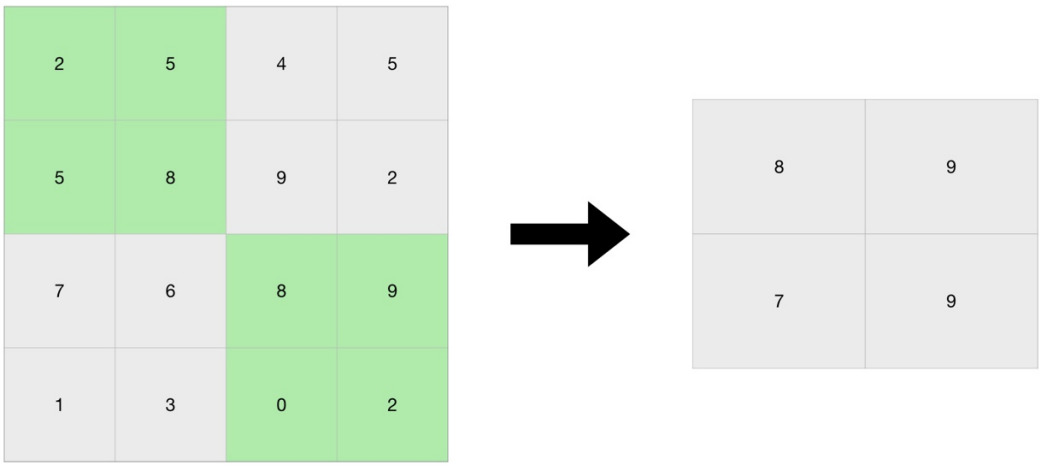
ReLU (Rectified Linear Units)

Varje faltningslager har också en aktiveringsfunktion. Den funktion som oftast används i neurala nätverk och faltningsnätverk kallas för ReLU som står för ”rectified linear units”. Funktionen kan definieras som $y = \max(0, x)$ och är linjär för alla positiva tal och noll för alla negativa tal. (Liu, 2017)

ReLU har använts som aktiveringsfunktion för alla faltningslager i nätverket dels för att den är de facto standard för faltningsnätverk och dels för att den kräver lite resurser vilket lämpar sig bra då nätverket ska klara av att tolka tecken i realtid.

Max-pooling

Efter varje faltningslager följer också det ett så kallat max-pooling lager. I det här fallet valdes det att använda max-pooling lager med dimensionen 2x2. Vad lagret gör är alltså att dela in bilden i 2x2 pixlars segment och sedan bara ta maxvärdet från de 4 pixlarna, illustration av hur max-pooling går till i figur 12.



Figur 12 Illustration av 2x2 max-pooling.

3.4.2 Träning

Alla modellerna tränas på samma data och itererar 10 gånger över träningsdatan. En iteration brukar kallas för en epok. Då man startar träningen specificerar man en kvot av träningsdatan som man vill att ska användas för validering, den datan används inte under

träningen utan används för validering i slutet av varje epok (i det här fallet användes 10% av träningsdatan som valideringsdata under träningen). Då kan man med hjälp av valideringsdata se huruvida modellen blir bättre på att klassificera data under träningens gång eller om den blir sämre. Ett säkert sätt att se att modellen blivit för van med träningsdatan är att träningsprecisionen ständigt stiger för varje epok medans valideringsprecisionen sjunker (fenomenet brukar på engelska kallas overfitting).

Förlustfunktion

Förlustfunktionen är en viktig del av modellen med vilken man kan se hur fel modellen har vid klassificering. För den här implementeringen har tensorflows inbyggda sparse categorical crossentropy förlustfunktion använts. Den lämpar sig för att användas på flerklassiga problem vilket det här problemet är eftersom det finns 26 klasser.

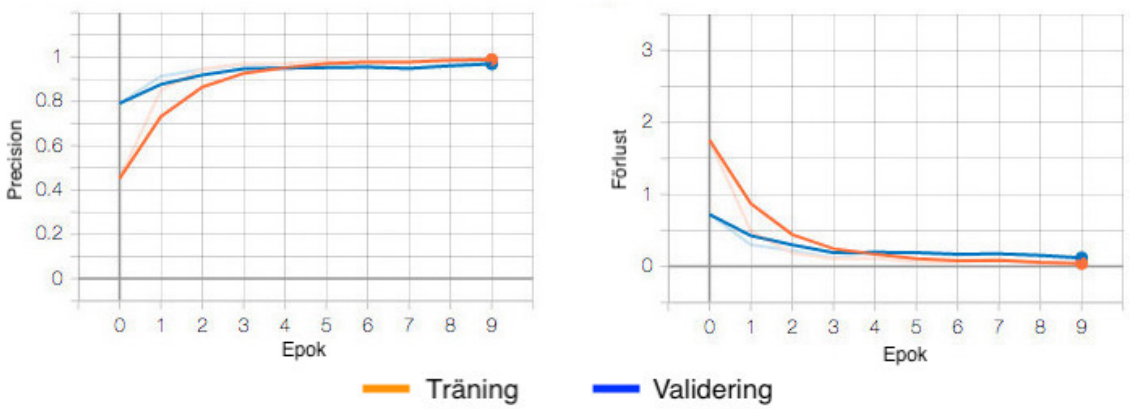
Optimering

Optimeringen är A och O för neurala närverk. Optimeraren är den del i systemet som uppdaterar alla parametrar beroende på förlusten. Optimerarens uppgift är att hitta rätt vikter för alla noder i nätverket för att uppnå en så låg förlust som möjligt. En optimeringsalgoritm som ofta används för faltningsnätverk är Adam. Adam är en resurseffektiv optimerare som lämpar sig bra för liknande problem som teckenklassificering. Av den anledningen har Adam också använts för den här implementeringen.

Dropout lager

Under träningen läggs också till ett så kallat dropout-lager till modellen. I det här fallet läggs det till det första faltningslagret med en dropout på 50% vilket betyder att hälften av aktiveringsnoderna avaktiveras varje gång något passerar genom lagret. Det här görs för att inte nätverket ska lära sig träningsdatan för bra och därigenom bli för van med den. Dropout används alltså bara under träningen. Vid evaluering används alla noder i lagret för att uppnå bästa resultat.

För att få en så bra modell som möjligt vill man alltså ha en modell vars valideringsförlust stadigt sjunker samtidigt som valideringsprecisionen stiger. Samtidigt bör man också hålla ett öga på att träningsprecisionen inte skiljer sig för mycket från valideringsprecisionen, eller att träningsprecisionen stiger samtidigt som valideringsförlusten gör det.



Figur 13 Tränings- och valideringsprecision och förlust för Canny_64 modellen, 40% utjämning har utförts på kurvorna för att jämna ut enstaka utstickare.

I figur 13 kan man se att Canny_64 modellens värden enligt de tidigare konstaterade kriterierna ser lovande ut. Valideringsprecisionen har en stigande kurva och förlusten en sjunkande. Träningsvärdena avviker inte heller märkbart från valideringsvärdena.

3.5 Resultat

För evaluering av modellerna används testdatan som skapades då korpusen delades dvs. 20% av hela korpusen. Eftersom testdatan kommer från samma korpus så är bilderna i den av liknande kvalitet och liknande bakgrunder. Därför bör man inte räkna med att uppleva exakt samma höga precision av modellerna om man skulle köra dem i en verklig situation där kameravinklar, kameralinser och liknande faktorer inverkar.

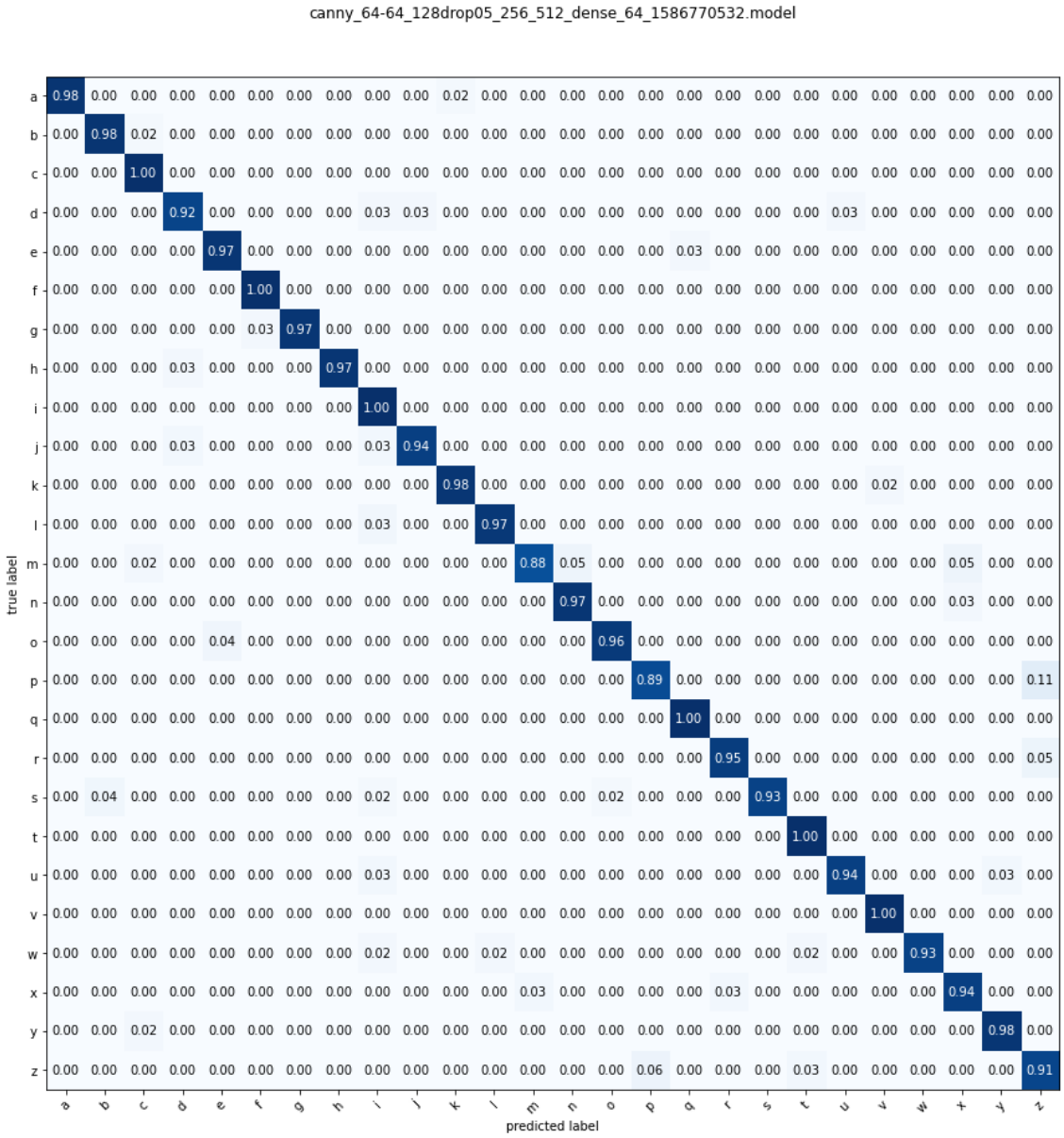
Den modellen som klarar sig bäst på testdatan är den som använder sig av kantupptäckning på en bild av storleken 64x64 pixlar. Den klarar av att klassificera tecknen med ca 96% precision vilket är en hel procentenhet bättre än den näst bästa som är den gråskaliga med en bild med dimensionerna 64x64.

Tabell 1 Jämförelse av modellerna

Modell	*Förlust*	*Precision*	*Tid (s)*
Canny_64	0,1686	0,9598	8,8
Gray_64	0,2164	0,9502	9,0
Gray_128	0,3537	0,9282	42,5
Canny_128	0,4059	0,9234	42,0

I tabell 1 kan man se att de modellerna som tar in en bild på 128x128 klarar sig betydligt sämre än modeller med 64x64. Dessutom tar det ca 4 gånger längre för 128 modellerna att klassificera testdatan. Den långsamma klassificeringstiden blir också synlig då man använder modellerna i realtid, användargränssnittet blir långsamt då videouppspelningen måste vänta på att varje bild ska klassificeras och videoströmmen blir hackig och oanvändbar på datorer med begränsad prestanda.

För det här ändamålet finns det alltså ingen nytta med att använda bilder större än 64x64 pixlar. Även om 128x128 bilderna innehåller mera information för modellerna att analysera så förvärras precisionen och det krävs mera resurser för att utföra klassificeringen.



Figur 14 Förväxlingsmatris för Canny_64 modellen (Confusion matrix).

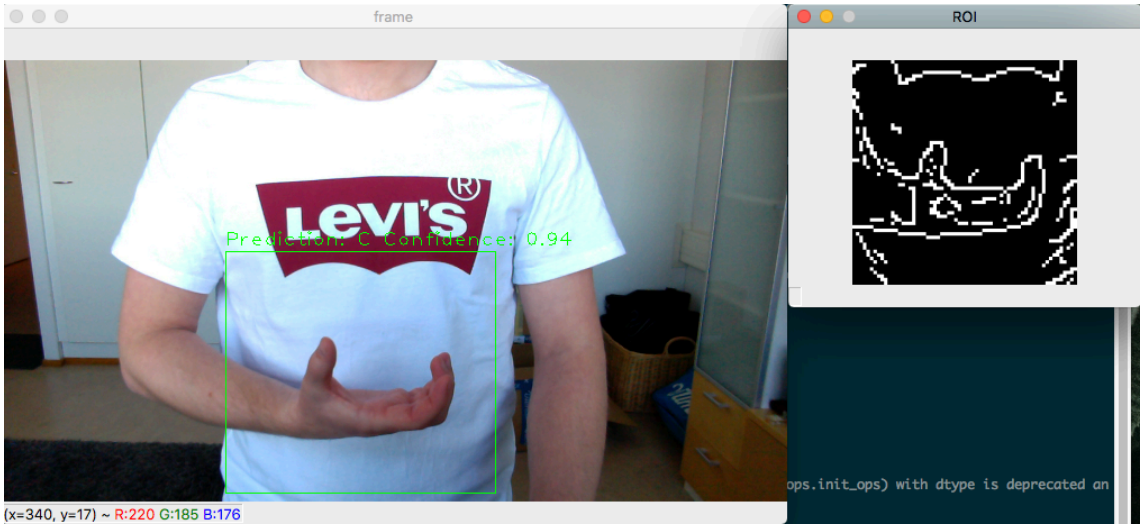
Förväxlingsmatrisen i figur 14 visar precisionen per bokstav för Canny_64 modellen. Man kan se att den ofta förväxlar P och Z vilket är föga förvånande med tanke på likheten på tecknen, se figur 15. M är den bokstav som modellen ändå har svårast med att klassificera men jämfört med P så är det inte bara en annan bokstav som modellen blandar ihop M med utan såväl C, N som X. Varför den tror att M är ett C 3% av tiden är svårförklarligt då det bara går åt ena hållet och modellen aldrig tror att ett C är ett M. Däremot gällande N och X kan man också se i figur 15 att de påminner en del om M. I figur 14 ser man också att modellen ibland kan förväxla N och X. Svårighet med att tolka bokstaven M är

en gemensam nämnare för alla modellerna och deras förväxlingsmatriser presenteras i bilaga.

Figur 15 Likheten mellan tecken som modellen förväxlar.

3.5.1 Tolken

Som en demonstration över att idén med att tolka teckenspråk på svenska i realtid är genomförbar har ett simpelt användargränssnitt byggts för att tolka tecken för en användare i realtid.



Figur 16 Tolken översätter bokstaven C.

Tolken är uppbyggd på liknande sätt som för datainsamlingen. Som konstaterats presenteras användaren med en ruta i midjehöjd i vilken användaren förväntas teckna. Hanteringen av videoströmmen har gjorts med OpenCV och inläsningen av modellerna har förstås gjorts med tensorflow eftersom de är byggda med hjälp av plattformen. För att modellerna ska klara av att klassificera tecknen bör den videoström den får ha undergått samma förbehandling som datan den tränats på. Det betyder att om Canny_64 modellen

används bör videoströmmen från intresseregionen förminskas och det bör utföras kant-upptäckning på varje frame. För att man ska få en bättre blick över vad modellen ser öppnas också en ruta med intresseregionen efter bildbehandlingen. I figur 16 ser man ett exempel på tolken som översätter bokstaven C med Canny_64 modellen.

4 SLUTDISKUSSION

Det här arbetet har behandlat tidigare forskning inom datorseende för ändamålet med teckenigenkänning. Baserat på den tidigare forskningen har sedan en implementering av en tolk för det svenska teckenspråket gjorts. Det har gjorts ett datorprogram för att generera data. En helt ny korpus för det svenska handalfabetet har skapats med hjälp av programmet. Även om det med hjälp av datainsamlingsprogrammet varit lättare att generera data har mycket tid lagts ner på att generera dessa bilder.

Modellen som skapades klarar av att klassificera testdatan med en precision på 96%. Det är en väldigt god precision och sannolikt beror på likheten mellan testdatan och träningsdatan. En så hög precision får man inte om man låter modellen tolka en YouTube video på någon som tecknar handalfabetet. Ändå klarar modellen av att i realtid översätta de tecken man visar framför den med en bra precision. Den precisionen är dock svår att mäta i och med att det också kan bero på att den som tecknar kanske inte tecknar rätt.

Planen är att i framtiden kunna implementera Canny_64 modellen eller en ännu bättre version på en webbsida som vem som helst ska ha åtkomst till. På det viset kunde det här arbetet hjälpa sådana som är i behov av tolkning. Ett användningsområde för modellen kunde vara att automatisera lärandet av handalfabetet. Då modellen kan säga om man tecknar rätt eller fel behöver inte en lärare kolla på alla tecken för att någon ska lära sig handalfabetet.

Jag hoppas att korpusen kommer att kunna användas för liknande implementeringar eller kunna inspirera till helt nya idéer för forskning som kan hjälpa icke tal-förmögna att kunna kommunicera även om den man kommunicerar med inte kan teckenspråk.

4.1 Vidare forskning

Ett förslag till vidare forskning är att implementera handigenkänning innan själva klassificeringsmodellen. På det viset kunde man få ner intresseområdet till ett mindre område där handen tar det mesta utrymmet. Då kunde man få en modell som blir bättre på att känna igen tecknen oavsett vilken hand eller vilken bakgrund som används. I och med det behöver man heller inte använda sig av ett statiskt intresseområde på skärmen utan man kan känna igen var på skärmen handen befinner sig. I och med det får användaren mer frihet och kan teckna på ett så naturligt sätt som möjligt. En utmaning med en handigenkänning innan själva klassificeringen av tecken skulle vara att få hela processen så resurseffektiv som möjligt så att även datorer med lägre prestanda klarar av att köra tolken.

Ett annat förslag kunde vara att använda sig av så kallad dataförstärkning (eng. data augmentation) för att öka antalet bilder som korpusen erbjuder och förändra dem så pass mycket att modellen blir bättre på att känna igen olika sorters händer och hudfärger.

Det här arbetet har uppnått något som aldrig gjorts förut, dvs. skapat en datorseendebaserad tolk för det svenska handalfabetet som klarar av att tolka i realtid. Jag hoppas att de resultat som producerats i någon mån kan vara till hjälp för någon som är i behov av en datorbaserad tolk eller någon som utvecklar något liknande hjälpmedel. All källkod för implementeringen är öppen och är tillgänglig på Lönnroth (2020). Där hittar man också korpusen.

KÄLLOR

Ilango, G. (6 april 2017). *Hand Gesture Recognition using Python and OpenCV - Part 1*. Hämtad mars 2020 från Gogul: https://gogul.dev/software/hand-gesture-recognition-p1

Institutet för språk och folkminnen. (16 maj 2014). *Svenskt teckenspråk*. Hämtad februari 2020 från isof.se: https://www.isof.se/om-oss/for-dig-i-skolan/sprak-for-dig-i-skolan/spraken-i-sverige/svenskt-teckensprak.html

Keskin, C., Kırac ̧ F., Kara, Y. E. & Akarun, L. (2012). *Hand Pose Estimation and Hand Shape Classification Using Multi-layered Randomized Decision Forests.* Istanbul: Bo ̆gazi ̧ci University, Computer Engineering Department.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Neural Information Processing Systems Conference.

Kuznetsova, A., Leal-Taixe ́, L. & Rosenhahn, B. (2013). *Real-time sign language recognition using a consumer depth camera*. 2013 IEEE International Conference on Computer Vision Workshops, 83-90.

Liu, D. (17 november 2017). *A Practical Guide to ReLU*. Hämtad april 2020 från Medium: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

Low vision international. (inget datum). *Synen - ett av våra viktigaste sinnen*. Hämtad februari 2020 från lvi.se: https://lvi.se/om-syn

Lönnroth, L. (2020). *Computer-visionbased-sign-language-interpretation*. Tillgänglig: https://github.com/LukasLonnroth/computer-visionbased-sign-language-interpretation

Mekala, P., Fan, J., Lai, W.-C. & Hsue, C.-W. (17 januari 2013). *Gesture Recognition Using Neural Networks Based on HW/SW Cosimulation Platform*. Hindawi Publishing Corporation.

Nagi, J. (2011). *Max-pooling convolutional neural networks for vision-based hand gesture recognition*.

Patterson, J. & Gibson, A. (2017). *Deep Learning, A Practitioner's Approach.* O'Reilly Media, Inc.

Pulkkis, G., Tana, J., Hellsten, T. & Karlsson, J. (2019). *Recent Developments in Computer Vision Based Real-Time Monitoring in Health and Well-Being.* Arcada Working Papers.

Rahaman, M. A., Jasim, M., Ali, M. & Hasanuzzaman, M. (2014). *Real-Time Computer Vision-Based Bengali Sign Language Recognition.* 2014 17th International Conference on Computer and Information Technology (ICCIT)*,* Dhaka, s. 192-197.

Rao, G. A. Saymala, K., Kishore, P. & Sastry, A. (2018). *Deep convolutional neural networks for sign language.* 2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*.* Vijaywada, s. 194-197.

Sangjun, O., Rammohan, M. & Minho, L. (2015). *Real Time Hand Gesture Recognition Using Random Forest and Linear Discriminant Analysis*.

Språkinstitutet. (inget datum). *Språk i Finland, Teckenspråk*. Hämtad Februari 2020 från Språkinstitutet: https://www.sprakinstitutet.fi/sv/om_sprak/sprak_i_finland/teckensprak

Stanford. (inget datum). *Convolutional Neural Network*. Hämtad April 2020 från Stanford.edu: http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

Sveriges Dövas Riksförbund. (2 augusti 2019). *Svenskt teckenspråk*. Hämtad februari 2020 från sdr.se: https://www.sdr.org/teckensprak

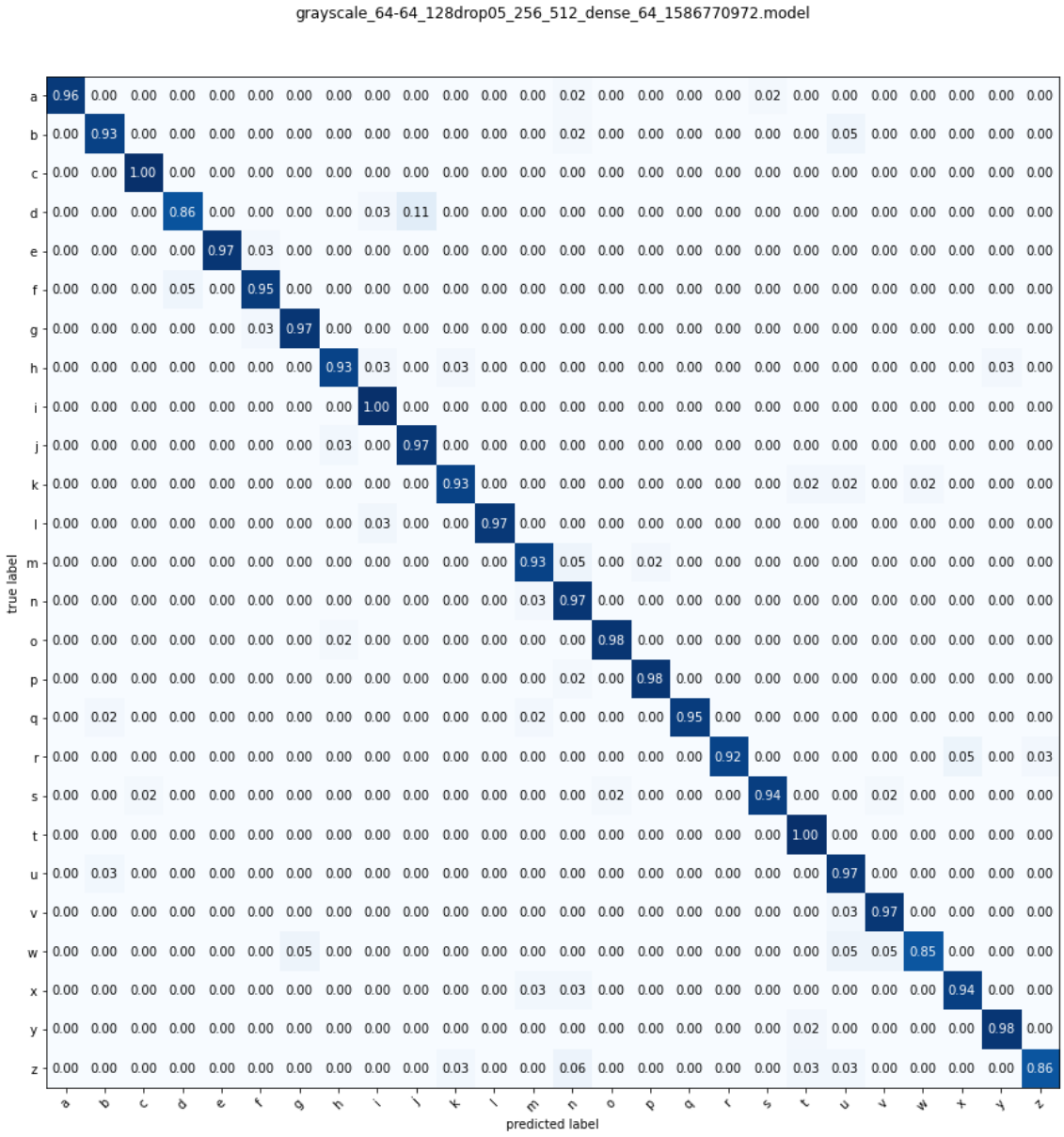
Viola, P. & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features.

Wang, W. & Pan, J. (2012). Hand segmentation using skin color and background information. *International Conference on Machine Learning and Cybernetics*.

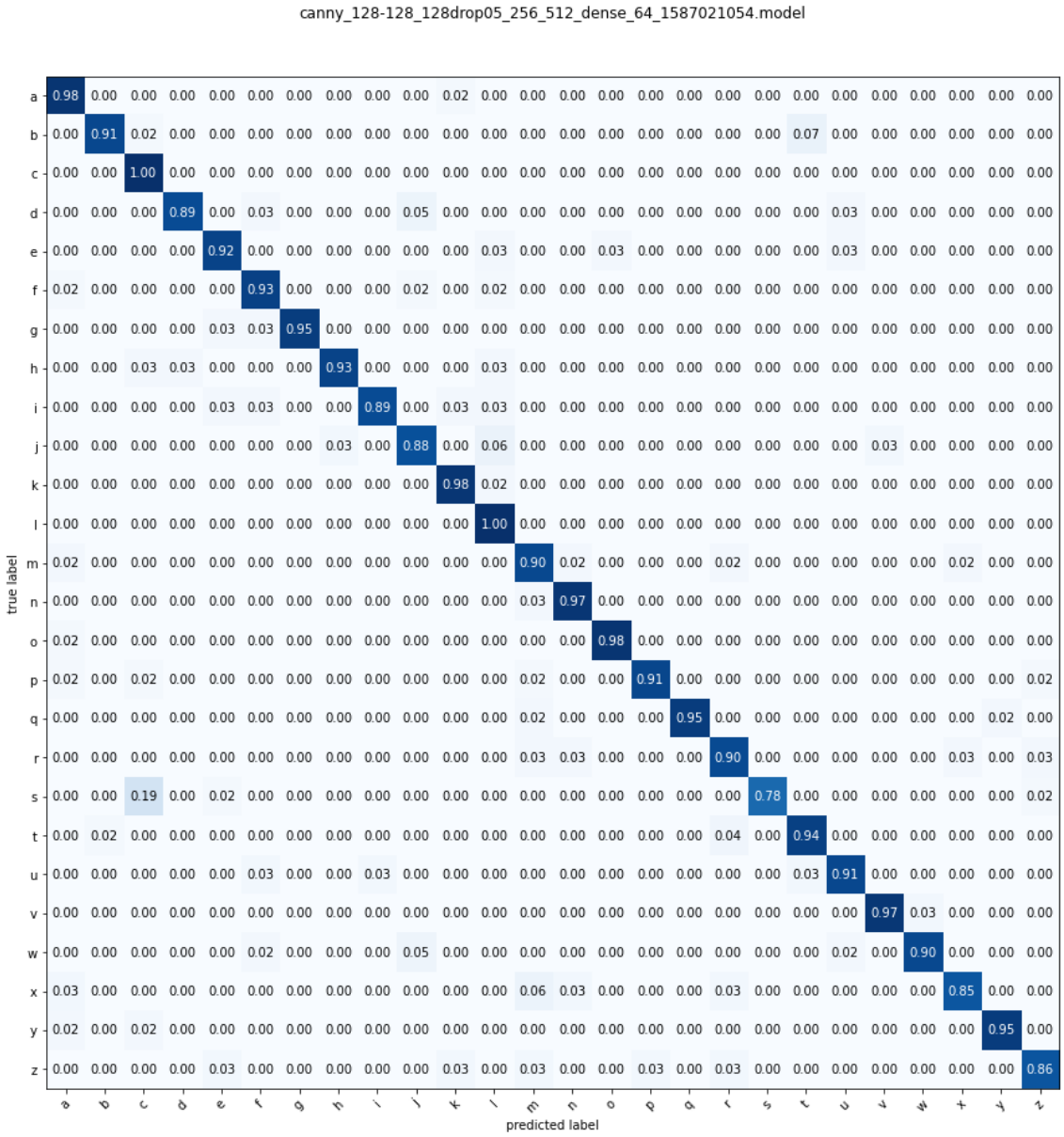
Yiu, T. (12 juni 2019). *Understanding Random Forest*. Hämtad mars 2020 från Towards Data Science: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

BILAGOR

Bilaga 1. Förväxlingsmatris för Gray_64



Bilaga 2. Förväxlingsmatris för Canny_128



Bilaga 3. Förväxlingsmatris för Gray_128